*The Embedded I/O Company*

TEWS TECHNOLOGIES

# TDRV005-SW-42

## VxWorks Device Driver

6 Channel SSI, Incremental Encoder, Counter

Version 5.0.x

## User Manual

Issue 5.0.1

July 2015

**TEWS TECHNOLOGIES GmbH**
Am Bahnhof 7 25469 Halstenbek, Germany
Phone: +49 (0) 4101 4058 0 Fax: +49 (0) 4101 4058 19
e-mail: info@tews.com www.tews.com

## TDRV005-SW-42

VxWorks Device Driver

6 Channel SSI, Incremental Encoder, Counter

Supported Modules:
TPMC117
TPMC317

This document contains information, which is proprietary to TEWS TECHNOLOGIES GmbH. Any reproduction without written permission is forbidden.

TEWS TECHNOLOGIES GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS TECHNOLOGIES GmbH reserves the right to change the product described in this document at any time without notice.

TEWS TECHNOLOGIES GmbH is not liable for any damage arising out of the application or use of the device described herein.

©2005-2015 by TEWS TECHNOLOGIES GmbH

| Issue | Description | Date |
|---|---|---|
| 1.0.0 | First Issue | December 14, 2005 |
| 1.0.1 | Examples and structures corrected | December 15, 2005 |
| 2.0.0 | New address TEWS LLC, general revision, user interface modified | July 17, 2008 |
| 2.1.0 | Support for TPMC317 added | February 12, 2009 |
| 3.0.0 | VxBus and SMP support added, initialization function tdrv005Init() added, address TEWS LLC removed | December 21, 2010 |
| 4.0.0 | VxWorks 6.9 64-Bit Support, new API Interface | January 13, 2012 |
| 5.0.0 | API Interface modified, Legacy functions removed | May 9, 2012 |
| 5.0.1 | List of recommended manuals modified | July 1, 2015 |

# Table of Contents


1 INTRODUCTION ..... 4

2 INSTALLATION ..... 5

2.1 Legacy vs. VxBus Driver ..... 6

2.2 VxBus Driver Installation ..... 7

2.2.1 Direct BSP Builds ..... 8

2.3 Legacy Driver Installation ..... 9

2.3.1 Include Device Driver in VxWorks Projects ..... 9

2.3.2 Special Installation for Intel x86 based Targets ..... 9

2.3.3 BSP Dependent Adjustments ..... 10

2.3.4 System Resource Requirement ..... 11

3 API DOCUMENTATION ..... 12

3.1 General Functions ..... 12

3.1.1 tdrv005Open ..... 12

3.1.2 tdrv005Close ..... 14

3.2 SSI Functions ..... 16

3.2.1 tdrv005SsiSetup ..... 16

3.2.2 tdrv005SsiRead ..... 19

3.3 Counter Functions ..... 21

3.3.1 tdrv005CounterSetup ..... 21

3.3.2 tdrv005CounterRead ..... 25

3.3.3 tdrv005CounterPreloadSet ..... 27

3.3.4 tdrv005CounterLoad ..... 29

3.3.5 tdrv005CounterReset ..... 31

3.3.6 tdrv005CounterWaitMatch ..... 33

3.3.7 tdrv005CounterWaitControlModeEvent ..... 35

3.4 Timer Functions ..... 37

3.4.1 tdrv005TimerSetup ..... 37

3.4.2 tdrv005TimerStart ..... 39

3.4.3 tdrv005TimerStop ..... 41

3.4.4 tdrv005TimerRead ..... 43

3.4.5 tdrv005TimerWait ..... 45

3.4.6 tdrv005TimerMultipleChannelReadSetup ..... 47

3.4.7 tdrv005TimerMultipleChannelReadWait ..... 49

3.5 Digital Input Functions ..... 52

3.5.1 tdrv005DigitalRead ..... 52

3.5.2 tdrv005DigitalWait ..... 54

3.6 Global Operation Functions ..... 56

3.6.1 tdrv005GlobalChannelEnable ..... 56

3.6.2 tdrv005GlobalChannelDisable ..... 58

3.6.3 tdrv005GlobalCounterPreloadSet ..... 60

3.6.4 tdrv005GlobalCounterLoad ..... 63

3.6.5 tdrv005GlobalMultipleChannelRead ..... 65

4 LEGACY I/O SYSTEM FUNCTIONS ..... 68

4.1 tdrv005PciInit ..... 68

# 1 Introduction

The TDRV005-SW-42 VxWorks device driver software allows the operation of the supported PMCs conforming to the VxWorks I/O system specification.

The TDRV005-SW-42 release contains independent driver sources for the old legacy (pre-VxBus) and the new VxBus-enabled driver model. The VxBus-enabled driver is recommended for new developments with later VxWorks 6.x release and mandatory for VxWorks 64-bit and SMP systems.

Both drivers, legacy and VxBus, share the same application programming interface (API) and device-independent basic I/O interface with open(), close() and ioctl() functions. The basic I/O interface is only for backward compatibility with existing applications and should not be used for new developments.

The TDRV005-SW-42 device driver and its API support the following features:

- operate channels in SSI mode
  - setup and configure channel (SSI or SSI listen-only)
  - read SSI data
- operate channels in Counter mode
  - setup and configure channel
  - read counter data
  - setup preload value
  - load preload value into counter
  - reset counter
  - wait for MATCH event
  - wait for ControlMode event
- operate the onboard interval timer
  - setup and configure interval timer
  - start and stop interval timer
  - read interval timer value
  - wait for interval timer event
  - setup and use interval timer as trigger event for simultaneous multiple channel read
- enable and disable multiple channels simultaneously
- simultaneously read multiple channel values
- setup and load counter preload values simultaneously

The TDRV005-SW-42 supports the modules listed below:

| | | |
|---|---|---|
| TPMC117 | 6 Channel SSI, Incremental Encoder, Counter | (PMC) |
| TPMC317 | 6 Channel SSI, Incremental Encoder, Counter | (PMC, Conduction Cooled |

**In this document all supported modules and devices will be called TDRV005. Specials for certain devices will be advised.**

To get more information about the features and use of TDRV005 devices it is recommended to read the manuals listed below.

| |
|---|
| TPMC117/TPMC317 User manual |

# 2 Installation

Following files are located on the distribution media:

Directory path ‘TDRV005-SW-42’:

| | |
|---|---|
| TDRV005-SW-42-5.0.1.pdf | PDF copy of this manual |
| TDRV005-SW-42-VXBUS.zip | Zip compressed archive with VxBus driver sources |
| TDRV005-SW-42-LEGACY.zip | Zip compressed archive with legacy driver sources |
| ChangeLog.txt | Release history |
| Release.txt | Release information |

The archive TDRV005-SW-42-VXBUS.zip contains the following files and directories:

Directory path ‘/tews/tdrv005’:

| | |
|---|---|
| tdrv005drv.c | TDRV005 device driver source |
| tdrv005def.h | TDRV005 driver include file |
| tdrv005.h | TDRV005 include file for driver and application |
| tdrv005api.h | TDRV005 API header file |
| tdrv005api.c | TDRV005 API file |
| Makefile | Driver Makefile |
| 40tdrv005.cdf | Component description file for VxWorks development tools |
| tdrv005.dc | Configuration stub file for direct BSP builds |
| tdrv005.dr | Configuration stub file for direct BSP builds |
| include/tvxbHal.h | Hardware dependent interface functions and definitions |
| apps/tdrv005exa.c | Example application |

The archive TDRV005-SW-42-LEGACY.zip contains the following files and directories:

Directory path ‘/tdrv005’:

| | |
|---|---|
| tdrv005drv.c | TDRV005 device driver source |
| tdrv005def.h | TDRV005 driver include file |
| tdrv005.h | TDRV005 include file for driver and application |
| tdrv005pci.c | TDRV005 device driver source for x86 based systems |
| tdrv005api.h | TDRV005 API header file |
| tdrv005api.c | TDRV005 API file |
| tdrv005exa.c | Example application |
| include/tdhal.h | Hardware dependent interface functions and definitions |

## 2.1 Legacy vs. VxBus Driver

In later VxWorks 6.x releases, the old VxWorks 5.x legacy device driver model was replaced by VxBus-enabled device drivers. Legacy device drivers are tightly coupled with the BSP and the board hardware. The VxBus infrastructure hides all BSP and hardware differences under a well defined interface, which improves the portability and reduces the configuration effort. A further advantage is the improved performance of API calls by using the method interface and bypassing the VxWorks basic I/O interface.

VxBus-enabled device drivers are the preferred driver interface for new developments.

The checklist below will help you to make a decision which driver model is suitable and possible for your application:

| Legacy Driver | VxBus Driver |
|---|---|
| ▪ VxWorks 5.x releases<br>▪ VxWorks 6.5 and earlier releases<br>▪ VxWorks 6.x releases without VxBus PCI bus support | ▪ VxWorks 6.6 and later releases with VxBus PCI bus<br>▪ SMP systems (only the VxBus driver is SMP safe)<br>▪ 64-bit systems (only the VxBus driver is 64-bit compatible) |

**TEWS TECHNOLOGIES recommends not using the VxBus Driver before VxWorks release 6.6. In previous releases required header files are missing and the support for 3rd-party drivers may not be available.**

## 2.2 VxBus Driver Installation

Because Wind River doesn't provide a standard installation method for 3rd party VxBus device drivers the installation procedure needs to be done manually.

In order to perform a manual installation extract all files from the archive TDRV005-SW-42-VXBUS.zip to the typical 3rd party directory *installDir/vxworks-6.x/target/3rdparty* (whereas *installDir* must be substituted by the VxWorks installation directory).

After successful installation the TDRV005 device driver is located in the vendor and driver-specific directory *installDir/vxworks-6.x/target/3rdparty/tews/tdrv005.*

At this point the TDRV005 driver is not configurable and cannot be included with the kernel configuration tool in a Wind River Workbench project. To make the driver configurable the driver library for the desired processor (CPU) and supported build tool (TOOL) must be built in the following way:

(1) Open a VxWorks development shell (e.g. C:\WindRiver\wrenv.exe -p vxworks-6.9)

(2) Change into the driver installation directory
*installDir/vxworks-6.x/target/3rdparty/tews/tdrv005*

(3) Invoke the build command for the required processor and build tool with optional VXBUILD argument
*make CPU=cpuName TOOL=tool [VXBUILD=xxx]*

For Windows hosts this may look like this:

```
> cd \WindRiver\vxworks-6.9\target\3rdparty\tews\tdrv005
> make CPU=PENTIUM4 TOOL=diab
```

To build SMP-enabled libraries, the argument VXBUILD=SMP must be added to the command line

```
> make CPU=PENTIUM4 TOOL=diab VXBUILD=SMP
```

To build 64-bit libraries, the argument VXBUILD=LP64 must be added to the command line

```
> make TOOL=gnu CPU=CORE VXBUILD=LP64
```

For 64-bit SMP-enabled libraries a build command may look like this

```
> make TOOL=gnu CPU=CORE VXBUILD="LP64 SMP"
```

To integrate the TDRV005 driver with the VxWorks development tools (Workbench), the component configuration file *40tdrv000.cdf* must be copied to the directory *installDir/vxworks-6.x/target/config/comps/VxWorks*.

```
C:> cd \WindRiver\vxworks-6.9\target\3rdparty\tews\tdrv005
C:> copy 40tdrv000.cdf \Windriver\vxworks-6.9\target\config\comps\vxWorks
```

In VxWorks 6.7 and newer releases the kernel configuration tool scans the CDF file automatically and updates the *CxrCat.txt* cache file to provide component parameter information for the kernel configuration tool as long as the timestamp of the copied CDF file is newer than the one of the *CxrCat.txt*. If your copy command preserves the timestamp, force to update the timestamp by a utility, such as *touch*.

In earlier VxWorks releases the CxrCat.txt file may not be updated automatically. In this case, remove or rename the original *CxrCat.txt* file and invoke the make command to force recreation of this file.

```
C:> cd \Windriver\vxworks-6.7\target\config\comps\vxWorks
C:> del CxrCat.txt
C:> make
```

After successful completion of all steps above and restart of the Wind River Workbench, the TDRV005 driver and API can be included in VxWorks projects by selecting the *"TEWS TDRV005 Driver" and "TEWS TDRV005 API"* components in the *"hardware (default) - Device Drivers"* folder with the kernel configuration tool.

### 2.2.1 Direct BSP Builds

In development scenarios with the direct BSP build method without using the Workbench or the vxprj command-line utility, the TDRV005 configuration stub files must be copied to the directory *installDir/vxworks-6.x/target/config/comps/src/hwif.* Afterwards the *vxbUsrCmdLine.c* file must be updated by invoking the appropriate make command.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tdrv005
C:> copy tdrv005.dc \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> copy tdrv005.dr \Windriver\vxworks-6.7\target\config\comps\src\hwif

C:> cd \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> make vxbUsrCmdLine.c
```

# 2.3 Legacy Driver Installation

## 2.3.1 Include Device Driver in VxWorks Projects

For including the TDRV005-SW-42 device driver into a VxWorks project (e.g. Tornado IDE or Workbench) follow the steps below:

(1) Extract all files from the archive TDRV005-SW-42-LEGACY.zip to your project directory.

(2) Add the device drivers C-files to your project.
Make a right click to your project in the 'Workspace' window and use the 'Add Files ...' topic.
A file select box appears, and the driver files in the tdrv005 directory can be selected.

(3) Now the driver is included in the project and will be built with the project.

**For a more detailed description of the project facility please refer to your VxWorks User's Guide (e.g. Tornado, Workbench, etc.)**

## 2.3.2 Special Installation for Intel x86 based Targets

The TDRV005 device driver is fully adapted for Intel x86 based targets. This is done by conditional compilation directives inside the source code and controlled by the VxWorks global defined macro **CPU_FAMILY**. If the content of this macro is equal to *I80X86* special Intel x86 conforming code and function calls will be included.

The second problem for Intel x86 based platforms can't be solved by conditional compilation directives. Due to the fact that some Intel x86 BSP's doesn't map PCI memory spaces of devices which are not used by the BSP, the required device memory spaces can't be accessed.

To solve this problem a MMU mapping entry has to be added for the required TDRV005 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tdrv005pci.c** contains the function *tdrv005PciInit().* This routine finds out all TDRV005 devices and adds MMU mapping entries for all used PCI memory spaces. Please insert a call to this function after the PCI initialization is done and prior to MMU initialization (*usrMmuInit()*).

The right place to call the function *tdrv005PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (it can be opened from the project *Files* window):

```
tdrv005PciInit();
```

Be sure that the function is called prior to MMU initialization otherwise the TDRV005 PCI spaces remains unmapped and an access fault occurs during driver initialization.

**Modifying the sysLib.c file will change the sysLib.c in the BSP path. Remember this for future projects and recompilations.**

### 2.3.3 BSP Dependent Adjustments

The driver includes a file called *include/tdhal.h* which contains functions and definitions for BSP adaptation. It may be necessary to modify them for BSP specific settings. Most settings can be made automatically by conditional compilation set by the BSP header files, but some settings must be configured manually. There are two way of modification, first you can change the *include/tdhal.h* and define the corresponding definition and its value, or you can do it, using the command line option *–D*.

There are 3 offset definitions (*USERDEFINED_MEM_OFFSET*, *USERDEFINED_IO_OFFSET*, and *USERDEFINED_LEV2VEC*) that must be configured if a corresponding warning message appears during compilation. These definitions always need values. Definition values can be assigned by command line option *-D<definition>=<value>*.

| Definition | Description |
|---|---|
| USERDEFINED_MEM_OFFSET | The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI memory space access |
| USERDEFINED_IO_OFFSET | The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI I/O space access |
| USERDEFINED_LEV2VEC | The value of this definition must be set to the difference of the interrupt vector (used to connect the ISR) and the interrupt level (stored to the PCI header ) |

Another definition allows a simple adaptation for BSPs that utilize a *pciIntConnect()* function to connect shared (PCI) interrupts. If this function is defined in the used BSP, the definition of *USERDEFINED_SEL_PCIINTCONNECT* should be enabled. The definition by command line option is made by *-D<definition>.*

**Please refer to the BSP documentation and header files to get information about the interrupt connection function and the required offset values.**

### 2.3.4 System Resource Requirement

The table gives an overview over the system resources that will be needed by the driver.

| Resource | Driver requirement | Devices requirement |
|---|---|---|
| Memory | < 1 KB | < 1 KB |
| Stack | < 1 KB | --- |
| Semaphores | 0 | 40 |

**Memory and Stack usage may differ from system to system, depending on the used compiler and its setup.**

The following formula shows the way to calculate the common requirements of the driver and devices.

*<total requirement> = <driver requirement> + (<number of devices> * <device requirement>)*

**The maximum usage of some resources is limited by adjustable parameters. If the application and driver exceed these limits, increase the according values in your project.**

# 3 API Documentation

## 3.1 General Functions

### 3.1.1 tdrv005Open

#### NAME

tdrv005Open – opens a device.

#### SYNOPSIS

```
TDRV005_HANDLE tdrv005Open
(
      char          *DeviceName
)
```

#### DESCRIPTION

Before I/O can be performed to a device, a device handle must be opened by a call to this function. If the legacy TDRV005 driver is used, this function will also install the legacy driver and create devices with the first call. The VxBus TDRV005 driver will be installed automatically by the VxBus system.

**The tdrv005Open function can be called multiple times (e.g. in different tasks)**

#### PARAMETERS

*DeviceName*

This parameter points to a null-terminated string that specifies the name of the device. The first TDRV005 device is named “/tdrv005/0” the second device is named “/tdrv005/1” and so on.

#### EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE hdl;

/*
** open the specified device
*/
hdl = tdrv005Open(“/tdrv005/0”);
```

```
if (hdl == NULL)
{
     /* handle open error */
}
```

## RETURNS

A device handle or NULL if the function fails. An error code will be stored in *errno*.

## ERROR CODES

The error codes are stored in *errno.*

The error code is a standard error code set by the I/O system.

## 3.1.2 tdrv005Close

### NAME

tdrv005Close – closes a device.

### SYNOPSIS

```
TDRV005_STATUS tdrv005Close
(
    TDRV005_HANDLE          hdl
)
```

### DESCRIPTION

This function closes previously opened devices.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

### EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE hdl;
TDRV005_STATUS result;

/*
** close the device
*/
result = tdrv005Close(hdl);

if (result != TDRV005_OK)
{
    /* handle close error */
}
```

## RETURNS

TDRV005_OK on success or an appropriate system error code.

## ERROR CODES

The error code is a standard error code set by the I/O system.

# 3.2 SSI Functions

## 3.2.1 tdrv005SsiSetup

### NAME

tdrv005SsiSetup – sets up a channel for SSI operation.

### SYNOPSIS

```
TDRV005_STATUS tdrv005SsiSetup
(
    TDRV005_HANDLE          hdl,
    unsigned char           Channel,
    TDRV005_SSI_SETUP       *Options
)
```

### DESCRIPTION

This function sets up a specific channel to the provided SSI configuration. The function returns immediately to the caller after setting up the corresponding channel.

**The SSI channel is not enabled by this command. This must be done by a subsequent call to tdrv005globalChannelEnable (see chapter 3.6.1)**

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*Channel*

This value specifies the channel which should be affected. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Only one channel may be specified.

*Options*

This value specifies the necessary configuration options in the structure *TDRV005_SSI_SETUP* with the following layout:

```
typedef struct
{
    TDRV005_SSI_MODE        Mode;
    unsigned char           NumberOfDataBits;
    TDRV005_SSI_CODING      Coding;
    unsigned char           ZeroBits;
    TDRV005_SSI_PARITY      Parity;
    unsigned char           ClockRate;
} TDRV005_SSI_SETUP;
```

## MEMBERS

*Mode*

This value specifies the desired operation mode of the SSI interface. Possible values are:

| Value | Description |
|---|---|
| TDRV005_MODE_STANDARD | Standard SSI operation |
| TDRV005_MODE_LISTENONLY | SSI interface operates in listen-only mode. |

*NumberOfDataBits*

This value specifies the number of data bits to use. Possible values are between 1 and 32.

*Coding*

This value specifies the desired coding format. Possible values are:

| Value | Description |
|---|---|
| TDRV005_CODING_BINARY | Binary coding is used. |
| TDRV005_CODING_GRAY | Gray coding is used, data is converted into binary. |

*ZeroBits*

This value specifies the number of zero bits to use in combination with parity. Possible values are either 0 or 1.

*Parity*

This value specifies what kind of parity bit should be used. Possible values are:

| Value | Description |
|---|---|
| TDRV005_PARITY_NONE | No parity bit is used. |
| TDRV005_PARITY_EVEN | An even parity bit is used. |
| TDRV005_PARITY_ODD | An odd parity bit is used. |

*ClockRate*

This value specifies the clock rate for the encoder's serial clock speed. The clock can be programmed in steps of 1µs in the range of 1 to 15.

## EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE     hdl;
TDRV005_STATUS     result;
TDRV005_SSI_SETUP  Options;

/*
** setup the counter with appropriate options
*/
Options.Mode                = TDRV005_MODE_STANDARD;
Options.NumberOfDataBits    = 32;
Options.Coding              = TDRV005_CODING_BINARY;
Options.ZeroBits            = 1;
Options.Parity              = TDRV005_PARITY_NONE;
Options.ClockRate           = 10;

result = tdrv005SsiSetup(hdl, TDRV005_CH0, &Options);

if (result != TDRV005_OK)
{
     /* handle configuration error */
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_INVAL | Invalid parameter specified. |

## 3.2.2 tdrv005SsiRead

### NAME

tdrv005SsiRead – reads the value of an SSI channel.

### SYNOPSIS

```
TDRV005_STATUS tdrv005SsiRead
(
    TDRV005_HANDLE      hdl,
    unsigned char       Channel,
    int                 Timeout,
    unsigned int        *Data,
    unsigned int        *Status
)
```

### DESCRIPTION

This function reads the value of the corresponding channel's data register. Only the number of previously configured data bits is valid. The function returns to the caller after the desired channel is read or the specified timeout occurred.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*Channel*

This value specifies the channel which should be affected. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Only one channel may be specified.

*Timeout*

This value specifies the timeout in milliseconds. If the function should wait indefinitely for the data to be valid, *TDRV005_WAIT_FOREVER* must be specified.

*Data*

This parameter points to an unsigned int value where the data register content is stored.

*Status*

This parameter points to an unsigned int value where the status register content is stored.

## EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE     hdl;
TDRV005_STATUS     result;
unsigned int       ssiValue;
unsigned int       ssiStatus;

/*
** read the current counter value
*/
result = tdrv005SsiRead(hdl,
                        TDRV005_CH0,
                        TDRV005_WAIT_FOREVER,
                        &ssiValue,
                        &ssiStatus);

if (result == TDRV005_OK)
{
     printf( “SSI Value  = 0x%08X\n”, ssiValue );
     printf( “SSI Status = 0x%08X\n”, ssiStatus );
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_INVAL | Invalid parameter specified. |
| TDRV005_ERR_BUSY | Channel is not configured to SSI mode, or there is another job in progress. |

# 3.3 Counter Functions

## 3.3.1 tdrv005CounterSetup

### NAME

tdrv005CounterSetup – sets up a channel for counter operation.

### SYNOPSIS

```
TDRV005_STATUS tdrv005CounterSetup
(
    TDRV005_HANDLE          hdl,
    unsigned char           Channel,
    TDRV005_COUNTER_SETUP   *Options
)
```

### DESCRIPTION

This function sets up a specific channel to the provided counter configuration. The function returns immediately to the caller after setting up the corresponding channel.

**The counter channel is not enabled by this command. This must be done by a subsequent call to tdrv005globalChannelEnable (see chapter 3.6.1)**

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*Channel*

This value specifies the channel which should be affected. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Only one channel may be specified.

*Options*

This value specifies the necessary configuration options in the structure *TDRV005_COUNTER_SETUP* with the following layout:

```
typedef struct
{
    unsigned char           Polarity;
    TDRV005_CNT_INPUT       InputMode;
    TDRV005_CNT_INDEX       IndexControlMode;
    TDRV005_CNT_SCM         SpecialCountMode;
    TDRV005_CNT_CLKDIV      ClockPrescaler;
} TDRV005_COUNTER_SETUP;
```

## MEMBERS

*Polarity*

This value specifies the input polarity of the specified channel. The Input Polarity Control can be used to adapt the input to the input source polarity of A, B and I. Use the following predefined values to generate an OR'ed polarity value.

| Value | Description |
|---|---|
| TDRV005_POLARITY_A_LOW | low-active signal, default is high-active |
| TDRV005_POLARITY_B_LOW | low-active signal, default is high-active |
| TDRV005_POLARITY_I_LOW | low-active signal, default is high-active |

*InputMode*

The Input Mode determines the input source and how the counter interprets these input signals. Possible values are:

| Value | Description | Input Source |
|---|---|---|
| TDRV005_INPUT_TIMER_UP | Timer mode Up | internal clock prescaler |
| TDRV005_INPUT_TIMER_DOWN | Timer mode Down | internal clock prescaler |
| TDRV005_INPUT_DIRECTION | Direction count | Input A & Input B |
| TDRV005_INPUT_UPDOWN | Up/Down count | Input A & Input B |
| TDRV005_INPUT_QUADRATURE_1X | Quadrature count 1x | Input A & Input B |
| TDRV005_INPUT_QUADRATURE_2X | Quadrature count 2x | Input A & Input B |
| TDRV005_INPUT_QUADRATURE_4X | Quadrature count 4x | Input A & Input B |

*IndexControlMode*

The Index Control Mode determines how the counter interprets events on the I-input. Possible values are:

| Value | Description |
|---|---|
| TDRV005_ICM_NO_INDEX_CONTROL | no I-control |
| TDRV005_ICM_LOAD_ON_INDEX | load on index signal |
| TDRV005_ICM_LATCH_ON_INDEX | latch on index signal |
| TDRV005_ICM_GATE_ON_INDEX | gate on index signal |
| TDRV005_ICM_RESET_ON_INDEX | reset on index signal |
| TDRV005_ICM_REFERENCE_MODE | reference mode<br>(quadrature input mode only) |
| TDRV005_ICM_AUTO_REFERENCE_MODE | auto-reference mode<br>(quadrature input mode only) |
| TDRV005_ICM_INDEX_MODE | index mode<br>(quadrature input mode only) |

*SpecialCountMode*

This value specifies the desired special count mode. Possible values are:

| Value | Description |
|---|---|
| TDRV005_SCM_CYCLING_COUNTER | No special count mode, cycling counter. |
| TDRV005_SCM_DIVIDE_BY_N | Divide-by-N mode. |
| TDRV005_SCM_SINGLE_CYCLE | Single cycle mode. |

*ClockPrescaler*

This value specifies the internal clock prescaler to be used. Possible values are:

| Value | Description |
|---|---|
| TDRV005_CLKDIV_1X | Prescaler 1x, 32 MHz clock |
| TDRV005_CLKDIV_2X | Prescaler 2x, 16 MHz clock |
| TDRV005_CLKDIV_4X | Prescaler 4x, 8 MHz clock |
| TDRV005_CLKDIV_8X | Prescaler 8x, 4 MHz clock |

## EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE          hdl;
TDRV005_STATUS          result;
TDRV005_COUNTER_SETUP   Options;

/*
** setup the counter with appropriate options
*/
Options.Polarity            = TDRV005_POLARITY_A_LOW |
                              TDRV005_POLARITY_B_LOW |
                              TDRV005_POLARITY_I_LOW;
Options.InputMode           = TDRV005_INPUT_UPDOWN;
Options.IndexControlMode    = TDRV005_ICM_NO_INDEX_CONTROL;
Options.SpecialCountMode    = TDRV005_SCM_CYCLING_COUNTER;
Options.ClockPrescaler      = TDRV005_CLKDIV_8X;

result = tdrv005CounterSetup(hdl, TDRV005_CH0, &Options);

if (result != TDRV005_OK)
{
     /* handle configuration error */
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_INVAL | Invalid parameter specified. |

## 3.3.2 tdrv005CounterRead

### NAME

tdrv005CounterRead – reads the value of a counter channel.

### SYNOPSIS

```
TDRV005_STATUS tdrv005CounterRead
(
    TDRV005_HANDLE      hdl,
    unsigned char       Channel,
    unsigned int        *Data,
    unsigned int        *Status
)
```

### DESCRIPTION

This function reads the value of the corresponding channel's data register. The function returns immediately to the caller.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*Channel*

This value specifies the channel on which the specified event should occur. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Only one channel may be specified.

*Data*

This parameter points to an unsigned int value where the data register content is stored.

*Status*

This parameter points to an unsigned int value where the status register content is stored.

## EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE     hdl;
TDRV005_STATUS     result;
unsigned int       CounterValue, Status;

/*
** read the current counter value
*/
result = tdrv005CounterRead( hdl,
                             TDRV005_CH0,
                             &CounterValue,
                             &Status);

if (result == TDRV005_OK)
{
     printf( “Counter Value = 0x%08X\n”, CounterValue );
     printf( “Status  Value = 0x%08X\n”, Status );
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_INVAL | Invalid parameter specified. |
| TDRV005_ERR_BUSY | Channel is not configured to counter mode. |

### 3.3.3 tdrv005CounterPreloadSet

#### NAME

tdrv005CounterPreloadSet – sets the preload register of a counter channel.

#### SYNOPSIS

```
TDRV005_STATUS tdrv005CounterPreloadSet
(
    TDRV005_HANDLE      hdl,
    unsigned char       Channel,
    unsigned int        PreloadValue
)
```

#### DESCRIPTION

Set the counter's preload register to the supplied value. The function returns immediately to the caller.

#### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*Channel*

This value specifies the channel which should be affected. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Only one channel may be specified.

*PreloadValue*

This value specifies the new value of the channel's preload register.

## EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE     hdl;
TDRV005_STATUS     result;
unsigned int       PreloadValue;

/*
** load counter value
*/
PreloadValue = 0x12345678;

result = tdrv005CounterPreloadSet(hdl, TDRV005_CH0, PreloadValue);

if (result != TDRV005_OK)
{
     /* handle error */
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_INVAL | Invalid parameter specified. |
| TDRV005_ERR_BUSY | Channel is not configured to counter mode. |

## 3.3.4 tdrv005CounterLoad

### NAME

tdrv005CounterLoad – loads the preload register into the counter channel.

### SYNOPSIS

```
TDRV005_STATUS tdrv005CounterLoad
(
    TDRV005_HANDLE      hdl,
    unsigned char       Channel
)
```

### DESCRIPTION

Load the counter to the previously specified value of the counter preload register. The function returns immediately to the caller. To simultaneously load multiple channels, please refer to chapter 3.6.4.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*Channel*

This value specifies the channel which should be affected. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Only one channel may be specified.

## EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE     hdl;
TDRV005_STATUS     result;

/*
** load counter value
*/
result = tdrv005CounterLoad(hdl, TDRV005_CH0);

if (result != TDRV005_OK)
{
     /* handle error */
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_INVAL | Invalid parameter specified. |
| TDRV005_ERR_BUSY | Channel is not configured to counter mode. |

## 3.3.5 tdrv005CounterReset

### NAME

tdrv005CounterReset – resets the counter channel.

### SYNOPSIS

```
TDRV005_STATUS tdrv005CounterReset
(
    TDRV005_HANDLE      hdl,
    unsigned char       Channel
)
```

### DESCRIPTION

Reset the counter value of the specified channel. The function returns immediately to the caller.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*Channel*

This value specifies the channel which should be affected. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Only one channel may be specified.

## EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE     hdl;
TDRV005_STATUS     result;

/*
** reset counter value
*/
result = tdrv005CounterReset(hdl, TDRV005_CH0);

if (result != TDRV005_OK)
{
     /* handle error */
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_INVAL | Invalid parameter specified. |
| TDRV005_ERR_BUSY | Channel is not configured to counter mode. |

## 3.3.6 tdrv005CounterWaitMatch

### NAME

tdrv005CounterWaitMatch – waits for a counter-match event.

### SYNOPSIS

```
TDRV005_STATUS tdrv005CounterWaitMatch
(
    TDRV005_HANDLE      hdl,
    unsigned char       Channel,
    unsigned int        CompareValue,
    int                 Timeout
)
```

### DESCRIPTION

Waits until the counter value matches the provided counter compare value. The function returns to the caller if the counter matches the provided compare value, or the specified timeout occurred.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*Channel*

This value specifies the channel on which the specified event should occur. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Only one channel may be specified.

*CompareValue*

This parameter specifies the value to which the counter should be compared.

*Timeout*

This value specifies the timeout in milliseconds. If the function should wait indefinitely for the event to occur, TDRV005_WAIT_FOREVER must be specified.

## EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE     hdl;
TDRV005_STATUS     result;
unsigned int       MatchValue;

/*
** wait indefinitely for counter match event
*/
MatchValue = 0x12345678;

result = tdrv005CounterWaitMatch(hdl,
                                 TDRV005_CH0,
                                 MatchValue,
                                 TDRV005_WAIT_FOREVER);

if (result != TDRV005_OK)
{
     /* handle error */
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_INVAL | Invalid parameter specified. |
| TDRV005_ERR_BUSY | Channel is not configured to counter mode, or a counter-match job is already pending. |
| TDRV005_ERR_TIMEOUT | The event has not occurred, timeout. |

## 3.3.7 tdrv005CounterWaitControlModeEvent

### NAME

tdrv005CounterWaitControlModeEvent – waits for a counter-control-mode event.

### SYNOPSIS

```
TDRV005_STATUS tdrv005CounterWaitControlModeEvent
(
    TDRV005_HANDLE      hdl,
    unsigned char       Channel,
    int                 Timeout
)
```

### DESCRIPTION

Wait for the control mode event of the counter. The function returns to the caller if the configured control mode event or the specified timeout occurred.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*Channel*

This value specifies the channel on which the specified event should occur. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Only one channel may be specified.

*Timeout*

This value specifies the timeout in milliseconds. If the function should wait indefinitely for the event to occur, TDRV005_WAIT_FOREVER must be specified.

## EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE     hdl;
TDRV005_STATUS     result;

/*
** wait indefinitely for counter control mode event
*/
result = tdrv005CounterWaitControlModeEvent(hdl,
                                            TDRV005_CH0,
                                            TDRV005_WAIT_FOREVER);

if (result != TDRV005_OK)
{
     /* handle error */
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_INVAL | Invalid parameter specified. |
| TDRV005_ERR_BUSY | Channel is not configured to counter mode, or a counter-match job is already pending. |
| TDRV005_ERR_TIMEOUT | The event has not occurred, timeout. |

# 3.4 Timer Functions

## 3.4.1 tdrv005TimerSetup

### NAME

tdrv005TimerSetup – sets up the timer.

### SYNOPSIS

```
TDRV005_STATUS tdrv005TimerSetup
(
    TDRV005_HANDLE          hdl,
    TDRV005_CNT_CLKFRQ      ClockFrequency,
    unsigned short          PreloadValue
)
```

### DESCRIPTION

This function sets up the onboard timer to the provided configuration. The function returns immediately to the caller. The interval timer remains stopped after a call to this function.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*ClockFrequency*

This value specifies the clock frequency used as clock source for the timer. Possible values are:

| Value | Description |
|---|---|
| TDRV005_CLKFRQ_1MHZ | 1 MHz clock |
| TDRV005_CLKFRQ_2MHZ | 2 MHz clock |
| TDRV005_CLKFRQ_4MHZ | 4 MHz clock |
| TDRV005_CLKFRQ_8MHZ | 8 MHz clock |

*PreloadValue*

This value specifies the preload value of the timer. If the interval timer is running, this value is loaded automatically every time the timer expires. The preload value is of 16bit width.

## EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE     hdl;
TDRV005_STATUS     result;

/*
** setup the interval timer with an interrupt frequency of 100 Hz
*/
result = tdrv005TimerSetup(  hdl,
                             TDRV005_CLKFRQ_1MHZ,
                             10000);

if (result != TDRV005_OK)
{
     /* handle error */
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_INVAL | Invalid parameter specified. |

## 3.4.2 tdrv005TimerStart

### NAME

tdrv005TimerStart – starts the timer.

### SYNOPSIS

```
TDRV005_STATUS tdrv005TimerStart
(
     TDRV005_HANDLE              hdl
)
```

### DESCRIPTION

Start the previously configured timer. The function returns immediately to the caller.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

### EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE     hdl;
TDRV005_STATUS     result;

/*
** start the previously configured interval timer
*/
result = tdrv005TimerStart(hdl);

if (result != TDRV005_OK)
{
     /* handle error */
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_NOT_CONFIG | The timer was not configured properly. |

### 3.4.3 tdrv005TimerStop

#### NAME

tdrv005TimerStop – stops the timer.

#### SYNOPSIS

```
TDRV005_STATUS tdrv005TimerStop
(
     TDRV005_HANDLE              hdl
)
```

#### DESCRIPTION

Stop the previously configured timer. The function returns immediately to the caller.

#### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

#### EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE     hdl;
TDRV005_STATUS     result;

/*
** stop the interval timer
*/
result = tdrv005TimerStop(hdl);

if (result != TDRV005_OK)
{
     /* handle error */
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_NOT_CONFIG | The timer was not configured properly. |

### 3.4.4 tdrv005TimerRead

#### NAME

tdrv005TimerRead – reads the current timer value.

#### SYNOPSIS

```
TDRV005_STATUS tdrv005TimerRead(
      TDRV005_HANDLE          hdl,
      unsigned short          *Data
)
```

#### DESCRIPTION

Read the current value of the timer. The function returns immediately to the caller.

#### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*Data*

This parameter points to an unsigned short value where the data register content is stored.

#### EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE     hdl;
TDRV005_STATUS     result;
unsigned short     TimerValue;

/*
** read the current interval timer value
*/
result = tdrv005TimerRead(hdl, &TimerValue);

if (result == TDRV005_OK)
{
     printf( “Timer Value = 0x%04X\n”, TimerValue );
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_INVAL | Invalid parameter specified. |

## 3.4.5 tdrv005TimerWait

### NAME

tdrv005TimerWait – waits for a timer event.

### SYNOPSIS

```
TDRV005_STATUS tdrv005TimerWait
(
    TDRV005_HANDLE      hdl,
    int                 Timeout
)
```

### DESCRIPTION

Wait for the timer event or the specified timeout to occur. The function returns to the caller if the timer has expired, or the specified timeout occurred.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*Timeout*

This value specifies the timeout in milliseconds. If the function should wait indefinitely for the event to occur, TDRV005_WAIT_FOREVER must be specified.

### EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE     hdl;
TDRV005_STATUS     result;

/*
** wait indefinitely for an interval timer event
*/
result = tdrv005TimerWait(hdl, TDRV005_WAIT_FOREVER);
if (result != TDRV005_OK)
{
     /* handle error */
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_NOT_CONFIG | The timer was not configured properly. |
| TDRV005_ERR_TIMEOUT | The timer event has not occurred, timeout. |
| TDRV005_ERR_NOT_RUNNING | The timer is not running. |

## 3.4.6 tdrv005TimerMultipleChannelReadSetup

### NAME

tdrv005TimerMultipleChannelReadSetupt – sets up channels for multiple read.

### SYNOPSIS

```
TDRV005_STATUS tdrv005TimerMultipleChannelReadSetup
(
    TDRV005_HANDLE      hdl,
    unsigned char       Channels
)
```

### DESCRIPTION

Configure specified channels for simultaneous sampling triggered by the timer. The function returns immediately to the caller.

**Channels configured to SSI listen-only mode may not be used with this function.**

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*Channels*

This value specifies the channels which should be read simultaneously. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Multiple channels may be OR'ed to one value.

## EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE     hdl;
TDRV005_STATUS     result;

/*
** setup channels 0+5 for simultaneous sampling triggered by timer
*/
result = tdrv005TimerMultipleChannelReadSetup( hdl,
                                               TDRV005_CH0 | TDRV005_CH5);

if (result != TDRV005_OK)
{
     /* handle error */
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_INVAL | Invalid parameter specified. |
| TDRV005_ERR_BUSY | Specified channel is busy with an SSI job. |

## 3.4.7 tdrv005TimerMultipleChannelReadWait

### NAME

tdrv005TimerMultipleChannelReadWait – waits for the simultaneous data timer event.

### SYNOPSIS

```
TDRV005_STATUS tdrv005TimerMultipleChannelReadWait
(
    TDRV005_HANDLE          hdl,
    TDRV005_MULTIPLEVALUES  *MultipleValues,
    unsigned int            *Timestamp,
    int                     *MoreDataAvailable
)
```

### DESCRIPTION

Return the values of simultaneously sampled channels. An array to store all channel values must be supplied to this function. The function waits for the timer event to occur on which the previously configured channels are sampled, or the timeout (1 second) occurred. Before using this function the timer must be configured properly.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*MultipleValues*

This is a pointer to a TDRV005_MULTIPLEVALUES structure, where the read values are stored. Note that only the previously configured channels return valid data, other data entries must be ignored. The TDRV005_MULTIPLEVALUES structure has the following layout:

```
typedef struct
{
    TDRV005_VALUE_BUF  Channel[6];
} TDRV005_MULTIPLEVALUES;
```

## MEMBERS

*Channel*

This parameter is an array of a TDRV005_VALUE_BUF structure which holds the returned channel data. The value for channel 0 is returned at array index 0, channel 5's value is located at array index 5. The TDRV005_VALUE_BUF structure has the following layout:

```
typedef struct
{
    unsigned int    Value;
    unsigned int    Status;
} TDRV005_VALUE_BUF;
```

## MEMBERS

*Value*

This parameter holds the returned channel value.

*Status*

This parameter holds the returned channel status.

*Timestamp*

This is a pointer to an unsigned int value where the number of occurred timer interrupts is returned.

*MoreDataAvailable*

This is a pointer to a boolean integer value. The value is TRUE if additional data is available. This might happen if the timer event triggering the multiple-channel-read operation appears too fast. An additional call to tdrv005timerMultipleChannelReadWait must be performed.

## EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE          hdl;
TDRV005_STATUS          result;
int                     MoreDataAvailable;
unsigned int            Timestamp;
TDRV005_MULTIPLEVALUES  MultipleValues;

/*
** read channels triggered by timer
*/
result = tdrv005TimerMultipleChannelReadWait(hdl,
                                             &MultipleValues,
                                             &Timestamp,
                                             &MoreDataAvailable);

if (result == TDRV005_OK)
{
     printf( “Timestamp  = %d\n”, Timestamp);
     printf( “Channel(0) = 0x%08X\n”, MultipleValues.Channel[0].Value);
     printf( “Channel(5) = 0x%08X\n”, MultipleValues.Channel[5].Value);
}
else
{
     /* handle error */
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_INVAL | Invalid parameter specified. |
| TDRV005_ERR_BUSY | Another job is already pending. |
| TDRV005_ERR_TIMEOUT | The timer event has not occurred, timeout. |
| TDRV005_ERR_NOT_CONFIG | The timer was not configured properly. |

# 3.5 Digital Input Functions

## 3.5.1 tdrv005DigitalRead

### NAME

tdrv005DigitalRead – reads the digital input lines.

### SYNOPSIS

```
TDRV005_STATUS tdrv005DigitalRead
(
    TDRV005_HANDLE      hdl,
    unsigned char       *Data
)
```

### DESCRIPTION

Read the current values of all digital input lines. The function returns immediately to the caller.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*Data*

This parameter points to an *unsigned char* value where the data register content is stored. Channel 0 is represented by bit 0, channel 5 is represented by bit 5 of the returned byte value.

## EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE     hdl;
TDRV005_STATUS     result;
unsigned char      DigitalValue;

/*
** read the current interval timer value
*/
result = tdrv005DigitalRead(hdl, &DigitalValue);

if (result == TDRV005_OK)
{
     printf( “Digital Value = 0x%02X\n”, DigitalValue );
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_INVAL | Invalid parameter specified. |

## 3.5.2 tdrv005DigitalWait

### NAME

tdrv005DigitalWait – waits for an event on a digital input line.

### SYNOPSIS

```
TDRV005_STATUS tdrv005DigitalWait
(
    TDRV005_HANDLE          hdl,
    unsigned char           Channel,
    TDRV005_TRANSITION      Transition,
    int                     Timeout
)
```

### DESCRIPTION

Wait for the specified transition (rising or falling edge) on the specified digital input line. The function returns to the caller if the specified transition or the specified timeout occurred.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*Channel*

This value specifies the channel on which the specified event should occur. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Only one channel may be specified.

*Transition*

This value specifies the transition on which the specified event should occur.

| Value | Description |
|---|---|
| TDRV005_TR_RISING_EDGE | Rising edge on digital input |
| TDRV005_TR_FALLING_EDGE | Falling edge on digital input |

*Timeout*

This value specifies the timeout in milliseconds. If the function should wait indefinitely for the event to occur, TDRV005_WAIT_FOREVER must be specified.

## EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE     hdl;
TDRV005_STATUS     result;

/*
** wait indefinitely for a rising edge on digital input of channel 1
*/
result = tdrv005DigitalWait(hdl,
                            TDRV005_CH1,
                            TDRV005_TR_RISING_EDGE,
                            TDRV005_WAIT_FOREVER);

if (result != TDRV005_OK)
{
     /* handle error */
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_INVAL | Invalid parameter specified. |
| TDRV005_ERR_TIMEOUT | The event has not occurred, timeout. |

# 3.6 Global Operation Functions

## 3.6.1 tdrv005GlobalChannelEnable

### NAME

tdrv005GlobalChannelEnable – globally enables multiple channels.

### SYNOPSIS

```
TDRV005_STATUS tdrv005GlobalChannelEnable
(
    TDRV005_HANDLE      hdl,
    unsigned char       Channels
)
```

### DESCRIPTION

Enable multiple channels (SSI or Counter) simultaneously. The function returns immediately to the caller.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*Channels*

This value specifies the channels which should be enabled. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Multiple channels may be OR'ed to one value.

## EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE     hdl;
TDRV005_STATUS     result;

/*
** enable channel 0 and channel 5
*/
result = tdrv005GlobalChannelEnable ( hdl,
                                      TDRV005_CH0 | TDRV005_CH5);

if (result != TDRV005_OK)
{
     /* handle error */
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_INVAL | Invalid parameter specified. |

## 3.6.2 tdrv005GlobalChannelDisable

### NAME

tdrv005GlobalChannelDisable – globally disables multiple channels.

### SYNOPSIS

```
TDRV005_STATUS tdrv005GlobalChannelDisable
(
    TDRV005_HANDLE      hdl,
    unsigned char       Channels
)
```

### DESCRIPTION

Disable multiple channels (SSI or Counter) simultaneously. The function returns immediately to the caller.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*Channels*

This value specifies the channels which should be disabled. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Multiple channels may be OR'ed to one value.

## EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE     hdl;
TDRV005_STATUS     result;

/*
** disable channel 0 and channel 5
*/
result = tdrv005GlobalChannelDisable (hdl,
                                      TDRV005_CH0 | TDRV005_CH5);

if (result != TDRV005_OK)
{
     /* handle error */
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_INVAL | Invalid parameter specified. |

## 3.6.3 tdrv005GlobalCounterPreloadSet

### NAME

tdrv005GlobalCounterPreloadSet – globally sets counter preload registers.

### SYNOPSIS

```
TDRV005_STATUS tdrv005GlobalCounterPreloadSet
(
    TDRV005_HANDLE          hdl,
    unsigned char           Channels,
    TDRV005_MULTIPLEVALUES  *MultipleValues
)
```

### DESCRIPTION

Perform a simultaneous setup of the preload registers of specified counter channels. The desired values must be supplied to this function as well as the channel numbers which are affected. The function returns immediately to the caller.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*Channels*

This value specifies the channels which should be preloaded. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Multiple channels may be OR'ed to one value.

*MultipleValues*

This is a pointer to a TDRV005_MULTIPLEVALUES structure, where the read values are stored. The TDRV005_MULTIPLEVALUES structure has the following layout:

```
typedef struct
{
    TDRV005_VALUE_BUF  Channel[6];
} TDRV005_MULTIPLEVALUES;
```

## MEMBERS

*Channel*

This parameter is an array of a TDRV005_VALUE_BUF structure which holds the preload data. The value for channel 0 is located at array index 0, channel 5's value is located at array index 5. The TDRV005_VALUE_BUF structure has the following layout:

```
typedef struct
{
    unsigned int    Value;
    unsigned int    Status;
} TDRV005_VALUE_BUF;
```

## MEMBERS

*Value*

This parameter holds the preload channel value.

*Status*

This parameter is not used for this function.

## EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE          hdl;
TDRV005_STATUS          result;
TDRV005_MULTIPLEVALUES  Values;

/*
** preload channel 0 and channel 5
*/
Values[0].Value = 0x00000000;
Values[5].Value = 0x50000000;

result = tdrv005GlobalCounterPreloadSet ( hdl,
                                          TDRV005_CH0 | TDRV005_CH5,
                                          Values);

if (result != TDRV005_OK)
{
     /* handle error */
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_INVAL | Invalid parameter specified. |
| TDRV005_ERR_BUSY | Channel is not configured to counter mode. |

## 3.6.4 tdrv005GlobalCounterLoad

### NAME

tdrv005GlobalCounterLoad – globally loads preload registers into counters.

### SYNOPSIS

```
TDRV005_STATUS tdrv005GlobalCounterLoad
(
    TDRV005_HANDLE      hdl,
    unsigned char       Channels
)
```

### DESCRIPTION

Perform a simultaneous preload of specified counter channels. The values stored in the corresponding preload registers are loaded into the specified counters simultaneously. The function returns immediately to the caller.

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*Channels*

This value specifies the channels which should be preloaded. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Multiple channels may be OR'ed to one value.

## EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE     hdl;
TDRV005_STATUS     result;

/*
** load channel 0 and channel 5
*/
result = tdrv005GlobalCounterLoad(hdl,
                                  TDRV005_CH0 | TDRV005_CH5);

if (result != TDRV005_OK)
{
     /* handle error */
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_INVAL | Invalid parameter specified. |
| TDRV005_ERR_BUSY | Channel is not configured to counter mode. |

## 3.6.5 tdrv005GlobalMultipleChannelRead

### NAME

tdrv005GlobalMultipleChannelRead – reads multiple channels simultaneously.

### SYNOPSIS

```
TDRV005_STATUS tdrv005GlobalMultipleChannelRead
(
    TDRV005_HANDLE          hdl,
    unsigned char           Channels,
    TDRV005_MULTIPLEVALUES  *MultipleValues
)
```

### DESCRIPTION

Return the values of simultaneously sampled channels. The desired channel numbers must be supplied to this function as well as an array to store all channel values. The function returns to the caller after the read operation is finished.

**Channels configured to SSI listen-only mode may not be used with this function.**

### PARAMETERS

*hdl*

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

*Channels*

This value specifies the channels which should be read simultaneously. The pre-defined values (TDRV005_CH0 – TDRV005_CH5) must be used. Multiple channels may be OR'ed to one value.

*MultipleValues*

This is a pointer to a TDRV005_MULTIPLEVALUES structure, where the read values are stored. The returned values are only valid for channels enabled by the parameter *Channels*. The TDRV005_MULTIPLEVALUES structure has the following layout:

```
typedef struct
{
        TDRV005_VALUE_BUF  Channel[6];
} TDRV005_MULTIPLEVALUES;
```

## MEMBERS

*Channel*

This parameter is an array of a TDRV005_VALUE_BUF structure which holds the returned channel data. The value for channel 0 is returned at array index 0, channel 5's value is located at array index 5. The TDRV005_VALUE_BUF structure has the following layout:

```
typedef struct
{
        unsigned int        Value;
        unsigned int        Status;
} TDRV005_VALUE_BUF;
```

## MEMBERS

*Value*

This parameter holds the returned channel value.

*Status*

This parameter holds the returned channel status.

## EXAMPLE

```
#include “tdrv005api.h”

TDRV005_HANDLE          hdl;
TDRV005_STATUS          result;
TDRV005_MULTIPLEVALUES  MultipleValues;

/*
** read channel 0 and channel 5 simultaneously
*/
result = tdrv005GlobalMultipleChannelRead( hdl,
                                           TDRV005_CH0 | TDRV005_CH5,
                                           &MultipleValues);

if (result == TDRV005_OK)
{
     printf(“Channel(0) = 0x%08X\n”, MultipleValues.Channel[0].Value);
     printf(“Channel(5) = 0x%08X\n”, MultipleValues.Channel[5].Value);
}
else
{
     /* handle error */
}
```

## RETURN VALUE

On success, TDRV005_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERRORS

| Error Code | Description |
|---|---|
| TDRV005_ERR_INVALID_HANDLE | The specified device handle is invalid. |
| TDRV005_ERR_INVAL | Invalid parameter specified. |
| TDRV005_ERR_BUSY | A MultipleChannelRead job is already pending, or a channel is busy with an SSI job. |
| TDRV005_ERR_TIMEOUT | Internal timeout. The values couldn’t be retrieved within 2 seconds. |

# 4 Legacy I/O System Functions

This chapter describes functions which are relevant only for the legacy TDRV005 driver.

## 4.1 tdrv005PciInit

### NAME

tdrv005PciInit – Generic PCI device initialization

### SYNOPSIS

void tdrv005PciInit()

### DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required PCI spaces (base address register) and to enable the TDRV005 device for access.

The global variable *tdrv005Status* obtains the result of the device initialization and can be polled later by the application before the driver will be installed.

| Value | Meaning |
|---|---|
| > 0 | Initialization successful completed. The value of tdrv005Status is equal to the number of mapped PCI spaces |
| 0 | No TDRV005 device found |
| < 0 | Initialization failed. The value of (tdrv005Status & 0xFF) is equal to the number of mapped spaces until the error occurs.<br>Possible cause: Too few entries for dynamic mappings in sysPhysMemDesc[].<br>Remedy: Add dummy entries as necessary (syslib.c). |

### EXAMPLE

```
extern void tdrv005PciInit();


tdrv005PciInit();
```